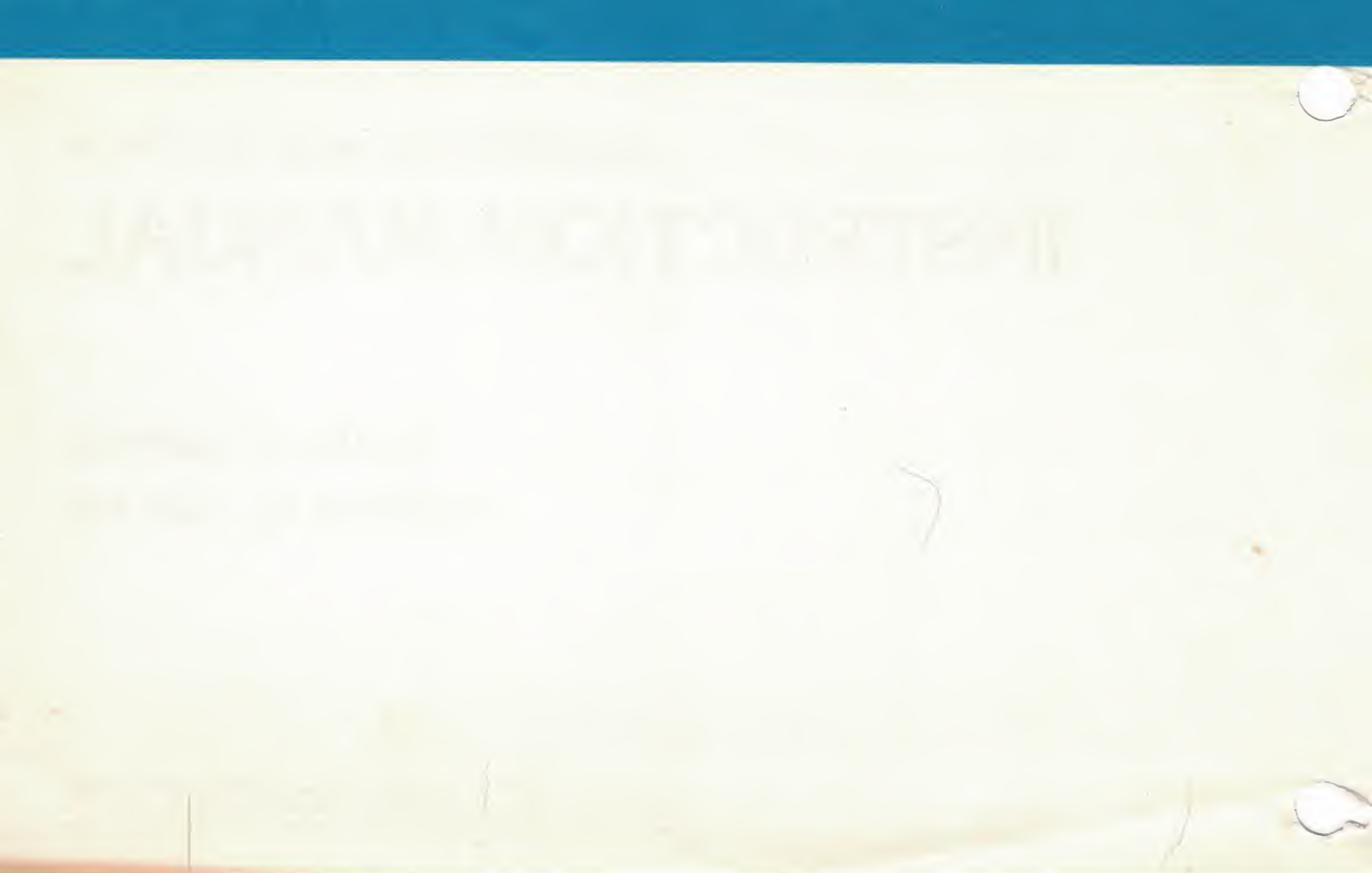


the ia7301 Computer in a Book

# INSTRUCTION MANUAL

by David Guzeman  
Published by Iasis, Inc.

© 1976, Iasis, Inc. All rights reserved. No part of this text or the programs contained herein may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form by any means, electronic, mechanical, optical, chemical, manual or otherwise without the prior written permission of Iasis Inc., 815 W. Maude Ave., Sunnyvale, CA 94086.



The obvious way to attack the problem of conveying to a reader an understanding of computer systems is to develop the concept of the computer as being made up of several basic parts. This concept is then expanded by developing each of the parts in turn until the system is representative of a real-world computer. There are sound reasons for this approach, indeed it is the one chosen for both the IAS ia515 Programmed Learning Course and the ia516 Microcomputer Applications Handbook. The authors of all these texts make one common assumption—the reader does not have a working computer system. All the information imparted to the reader must therefore take place within the confines of the printed page.

Which brings us to the case in point. The reader of the ia7301 book has in his possession a working computer that he can use to try theories, illustrate points, debug programs, etc. The computer thus becomes as much a part of the information transfer process as does the book itself.

This fact has caused us to treat the subject of computers in a completely different manner. Less emphasis is placed on building a paper model of a computer; instead the actual computer is presented right at the beginning and the reader is immersed in it to solve problems. A feeling for "what's inside the computer" comes with the understanding of how to use it.

If the reader has ever been exposed to a computer programming course that taught FORTRAN or BASIC, he is in for a rude shock. These "languages" will work on any large or medium scale computer, and as long as the memory capacity of the computer is not exceeded the user will not be aware of any differences from system to system. The user's program will run on any of these computers equipped for that level of language.

It is very easy to overlook the fact that there is nothing inherent in the design of the computer that allows the system to understand even English, let alone BASIC or Fortran. That ability resides in the computer only because special high-level language "compiler" programs have been loaded into the computer. These programs interpret statements written in BASIC and direct the computer in how to execute them. Simple arithmetic operations, like divide, that we take wholly for granted may require hundreds of machine operations.

Because high-level languages are extremely wasteful of memory they have not been very popular with microcomputer users. Applications handled by a program written in BASIC are fine for situations requiring only one computer, but the excessive memory requirements make this section impractical as a production model. Thus the microcomputer user must work directly in machine language where simple instructions like multiplication and division do not exist. It is up to the user to build his own multiplication and division routines out of machine instructions. Needless to say this requires a tremendous amount of understanding on the part of the user.

So let's begin.

Congratulations! You are the proud owner of the ia7301, the world's first Computer-in-a-Book. If you will follow along with us now, we'll give you a quick guided tour of the computer. Don't be alarmed if we seem to skip some of the buttons or features. They'll be covered at the time they're first needed. Right now, we merely want to acquaint you with enough of the system to get you into your first program.

**Applying Power to the System.** Before the computer can be used it must have power applied through the edge connector fingers. The ia7301 uses 28 fingers at the top of the board to make the power, I/O and tape interface connections. The fingers are repeated on both sides of the board so that any given connection can be made to the system on either the finger on top of the board or the corresponding finger on the bottom. To facilitate the process of making connections to the system, a 28-contact double read-out edge connector is provided. Wires can be soldered or clipped to the solder eyelets on the connector which is then slid onto the PC board so that contact is made to the fingers on the board.

Since the fingers on top of the board are electrically connected to the corresponding fingers on the bottom of the board through traces and components on the board itself, we can make connections to the system by using either row of eyelets on the connector. The power connections to the system must carry much more current than the tape or I/O contacts. To prevent any one contact from carrying excessive current, which would burn it, four contacts are used for each of the power connections. This distributes the current over four sets of contacts and fingers and eliminates the possibility of destroying the contacts.

It will be necessary for you to solder wire jumpers onto the connector to make the power connections. Jumpers should be soldered between contacts 1-2, 3-4, 5-6, 1-A, 2-B, 3-C, 4-D, 5-E and 6-F. The black stranded cable should then be soldered to the 1-2-A-B contacts, the white stranded cable to the 3-4-C-D contacts and the red stranded cable to the 5-6-E-F contacts. Connect the other end of the white cable to a +12V supply, the red cable to a +5V supply and the black cable to the ground return for the supplies. See Figs 1-1 through 1-6 for the power connections and wiring diagram.

Turn the system on by energizing the power supply, either by plugging it in, or if it has an ON-OFF switch, by flipping the switch to the ON position.

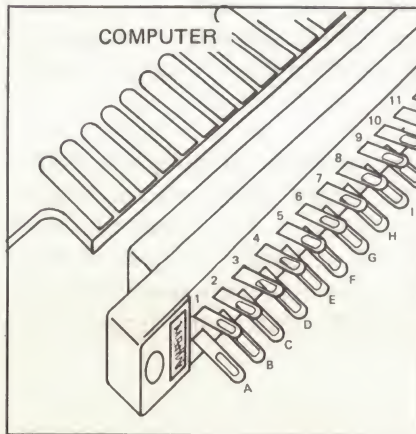


Fig. 1-1. The edge connector used with the ia7301 is a 28/56 double readout type. This means contact is made to 28 fingers on the bottom of the board. Solder eyelets are provided on the back of the connector for wires. The top row of eyelets is numbered 1-28 and the bottom row A-F.

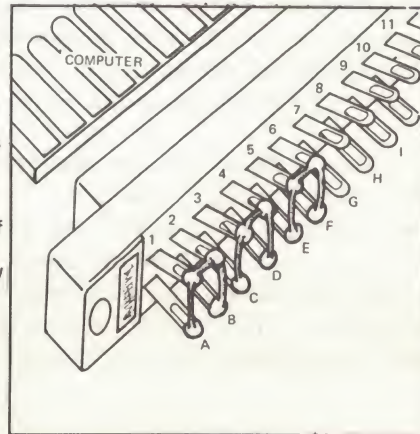
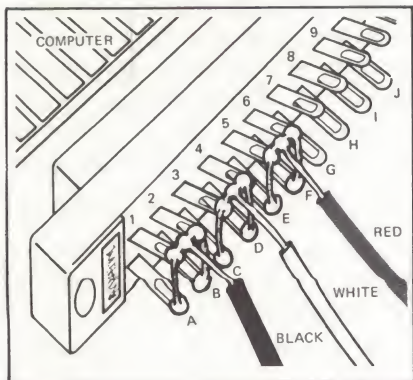
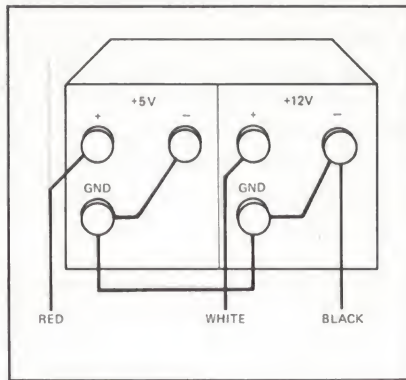


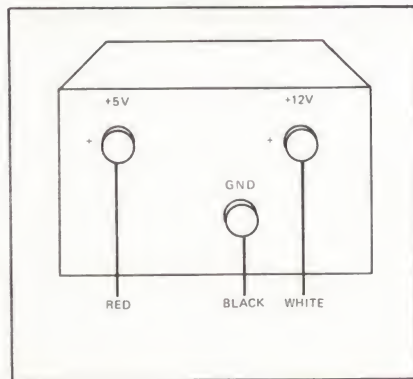
Fig. 1-2. To make power connections to the system, it will be necessary to solder jumper wires on the 1-2-A-B; 3-4-C-D; and 5-6-E-F eyelets using wire provided.



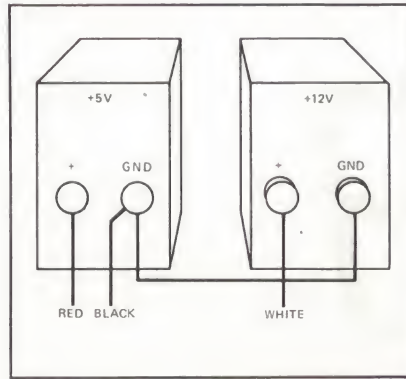
**Fig. 1-3.** Solder power cables onto the jumper wires using the cables provided. The black cable should be soldered to the jumper on the 1-2-A-B eyelets; the white cable on the 3-4-C-D eyelets and the red cable to the 5-6-E-F eyelets.



**Fig. 1-4.** Connection diagram for dual voltage power supply with separate - and GND (ground) connections.



**Fig. 1-5.** Connection diagram for dual voltage power supply with common GND.



**Fig 1-6.** Connection diagram for separate power supplies

**The Program.** Most digital systems today are characterized by a set of digital logic elements wired together to operate on a set of input signals and produce output signals used to perform a useful task. Thus, an industrial control system might use microswitch closures for inputs and drive solenoid valves at its outputs. The digital system for performing this function is dedicated in the sense that it is, to a large extent, unchangeable. The system built for controlling operations in an oil refinery cannot be used as the digital system in an oscilloscope. The two systems are totally different; each must be designed for its own kind of application and will be unique to that application.

The computer attacks the problem with a set of general-purpose circuits that are the same no matter which application is used. The computer is tailored to the specific application by the program that it contains. While there may be slight differences from one system to the next in the same way and perform the same functions within the system. In this way the computer resembles the human mind which can also address itself to a wide range of applications changing

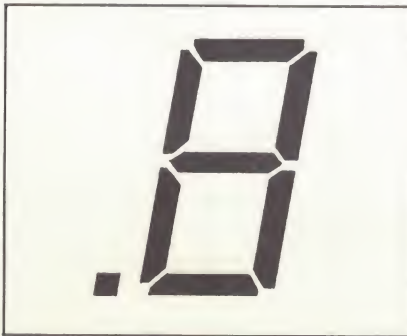


Fig. 1-7. Seven segment displays refer to numeric displays built of seven LED segments. By energizing certain of the segments, numbers and letters can be made to light up.

Actually the term "seven-segment" is something of a misnomer since including the decimal point brings the total to eight.

only the thought processes which lead to the solution. The program in the computer functions like a train of thought; it directs the computer to analyze the problem or perform the desired task. We can think of the program, therefore as the instructions which cause the computer to do something or solve a problem.

The instructions that make-up the program must be arranged in a meaningful sequence or pattern so that the computer will perform correctly. In this sense, the instructions are like words. Random words form no pattern, but when arranged correctly a sonnet results. By correctly arranging the instructions for the computer we can cause it to do most anything we want. The act of combining instructions in the pattern necessary to make the computer do our bidding is called programming.

Of course, it is not enough to write the programs on a piece of paper. They must somehow be communicated to the computer so that it will be able to follow the program. This is done by storing the program electronically in the system memory. Here the computer can retrieve the instructions one at a time in the correct order and execute them to perform the desired task; the memory stores the program in the same way our pad of paper stored the program while we were writing it. Unfortunately the very process of placing the instructions in memory is a complex one. The computer itself must be used to load the instructions into memory; the normal method is to use a keyboard for expressing the instructions. The computer reads the keyboard as the operator is writing the program and causes the instructions to be loaded into the memory. The very act of reading the keyboard and loading the memory requires that the computer be properly instructed in the loading process. The problem is akin to teaching a human to read so that he can understand the sonnet. This is done by a special program called the monitor program which is permanently built into the computer. When the computer is first turned on or the CLR , CLEAR key pressed, the computer begins to read and execute the monitor program.

The first operation performed by the computer under the direction of the monitor program is to check the keys in the keyboard to see if any of them are closed. In fact, it performs this checking process continually until it finds a key closure. Thus the monitor causes the computer to sit and wait for instructions from its human operator. These instructions will come in the form of electrical signals from the keyboard. As these arrive the computer will check them to see which operation is desired. One of the most common operations of all is to load the memory with a set of instructions which make up a program the operator wishes performed or executed. For example, the operator may use the keyboard to enter a program for balancing his checkbook. When the program is completely stored in memory, depressing **EXC** , EXECUTE, will cause the computer to execute the checkbook program and return to the monitor mode when completed.

**Entering Programs in the ia7301.** Your ia7301 contains a monitor program, a keyboard for entering programs and a memory for storing programs. In addition it contains a set of eight 7-segment displays for displaying instructions and results of programs. Thus the displays can be used to monitor the instructions that are being loaded into memory so that keyboard errors are quickly detected and corrected. Then when the program is executed the same displays will contain the results generated by the computer, for example, the correct checkbook balance. Of course, we need to tell the computer when the keys are being used to enter programs. We do that by using one special key, **DCM** , Display and Change Memory. When this key is depressed the computer immediately knows that any key entries that follow are to be interpreted as instructions to be loaded into memory. DCM is the mode we will use for entering programs into memory.

When the power is first applied to the system, a special reset pulse is generated that causes the system to come up in the system monitor program. This will allow you to begin entering and executing programs immediately. Entry into the system is therefore as close as the keyboard. You can tell if the system came up in the monitor program correctly by checking the LED displays. All eight of the displays should have a number or other symbol in them. The monitor program is written so that the displays should show the DCM symbol in the left-most display digit and dashes in the other seven. You can recognize the DCM symbol because it looks like the lower half of a football goalpost.

Energizing the system frequently causes the system to go right to the DCM mode. Even if DCM is not entered, the monitor program should still be in control of the system. In this case the displays are lighted but do not contain the DCM symbol and dashes. This is nothing to worry about; just press **DCM** and the system will correct the displays.

When the DCM symbol and dashes appear on the displays, it is your sign that the system has correctly entered the DCM mode. DCM stands for Display and Change Memory. This is the most basic of the monitor modes and is entered automatically from some of the other modes. It is also automatically entered whenever the **CLR** key is pressed.

You can see the automatic DCM feature by pressing one of the other mode keys, **DCR** . We will discuss this mode in a moment. For now just check to be sure the DCR symbol appears (the top half of the football goal post) followed by an A, a few dashes and some letters and numbers. If you now press **DCM** the system will automatically return to DCM displaying the DCM symbol and a set of seven dashes.

Enter:

**DCR**

See Displayed:

**U A - - - - - \***

\* **■** is used to denote a symbol, letter or number, that is randomly generated and cannot be predetermined.

**DCM**

**n - - - - -**

We can accomplish the same thing by depressing **CLR** for clear.

Enter:

**DCR**

See Displayed:

**U A - - - - -**

**CLR**

**n - - - - -**

**Addressing Memory.** The memory of the computer is used for storing programs. It will also be used for storing the data that will be used by the program. Thus the memory will store both the program for balancing our checkbook and the list of outstanding checks and their amounts. We will cover the mechanism of how the computer keeps the program and the data separated later. Now we only need to know that the memory can store data and program instructions.

The memory is made up of many separate locations, each capable of storing one thing. We can think of the system memory as a set of post office boxes. To distinguish one box from the next we will assign each one an address. This allows us to conveniently keep track of where various things are stored in the memory. Since each box or location is assigned an address we need only tell the computer that the program we wish to execute begins at, say, location 100 and the system will then know where to find the first instruction of our program. Without these addresses the memory would be a hopeless jumble of random data and programs, and there would be no way to fetch from the memory.

We can see these addresses by using the DCM mode.

Enter:	See Displayed:
DCM .	n - - - - -
0 0 0 0	n 0 0 0 0 - - -

When we press **DCM** the computer responds by displaying the DCM mode symbol and seven dashes. It is now waiting for an address. In the example above we have keyed in address 0000. Notice that each time we press the 0 key, one of the dashes is replaced with a 0. The actual process is one of shifting in the keyed numeral from the right; each numeral in the display is shifted one place to the left to make room for the new entry. This can be seen by entering some other digits, 0-5.

Enter:

1  
2  
3  
4  
5  
0 0 0 0

See Displayed:

n 0 0 0 0 - - -  
n 0 0 0 1 - - -  
n 0 0 1 2 - - -  
n 0 1 2 3 - - -  
n 1 2 3 4 - - -  
n 2 3 4 5 - - -  
n 0 0 0 0 - - -

Notice that the DCM mode defines a 4-digit address field in the displays; an address always consists of four digits. We will need to specify the various digits in the displays frequently so it will greatly facilitate our discussions if we establish a system for referring to the digits. Taking the right-most display digit as  $D_0$  through  $D_7$ , see Fig 1-8.

Using the digit identification system we can see that the DCM mode treats digits  $D_6$ ,  $D_5$ ,  $D_4$  and  $D_3$  as the address field of the display. As new numbers are entered through the keyboard they appear at  $D_3$ . The number that was already at  $D_3$  is shifted to  $D_4$ , the number at  $D_4$  to  $D_5$ , the number at  $D_5$  to  $D_6$ . The number that was at  $D_6$  is not shifted into  $D_7$ ; to do that would erase the DCM mode symbol. Instead the original number at  $D_6$  is shifted out of the system completely and is lost.

This shifting process allows errors to be easily corrected by simply keying in the correct address. As each numeral is entered it displaces the original one and moves the digit to the left; when the last digit is entered the new four digit address will have completely displaced the old one.

**IDIOSYNCRASY:** THE ia7301 DOES NOT RECOGNIZE DASHES AS A VALID PART OF AN ADDRESS. THEREFORE, WHEN ENTERING THE ADDRESS 0100 INTO THE SYSTEM YOU MUST BE SURE TO INCLUDE THE LEADING ZERO OR AN ERROR WILL RESULT!

When you are satisfied with the address as displayed, inform the system by pressing **NXT** , NEXT. This is a sign to the system that it should treat the data on the displays as a valid address and fetch the contents of that memory location to the displays for you to examine. In the case of address 0000 there will be a BF stored in that location. Each location holds two and only two digits. Notice where they appear and notice that a dash separates the address location from its contents.

Enter:

DCM

NXT

See Displayed:

n00000--

n00000-bf

By keying in an address and then pressing **NXT** we can examine any memory location in the system, including those that contain the system monitor. We will frequently want to examine a whole set of locations to check an entire program. To have to key in an address for each of the locations would be very tedious and error prone. The ia7301 covers this possibility by automatically incrementing the address each time **NXT** is pressed.

Enter:

DCM 0 0 0 0 NXT

NXT

NXT

NXT

NXT

See Displayed:

n0000-bF

n0001-83

n0002-bF

n0003-A4

n0004-C0

Pressing **NXT** the first time serves to tell the computer that you are satisfied with the address as it appears on the displays. The computer responds by showing you the contents of that memory location. Further depressions of **NXT** will increment the address as illustrated above. If you need to jump to a different sequence of addresses, this can be accomplished by pressing **DCM** and keying in the initial address of the next sequence.

Enter:

NXT

DCM

8 0 0 0

NXT

NXT

See Displayed:

n0004-C0

n0005-C0

n-----

n8000---

n8000-31

n8001-00

**Number Systems.** So far we have completely ignored the question of how to express the data and instructions in the computer. Some of the examples used so far, 0004, have been familiar while others, BF, have, undoubtedly, not been familiar. The key to expressing addresses and data to the computer is in understanding that regardless of how we write and talk about numbers, the computer handles numbers in binary.

As human beings we are accustomed to using the decimal number system in our day to day dealings. This system is based upon the use of ten symbols or numerals to express numbers. The symbols chosen are 0 through 9 and with only one digit or "place" this is the range of numbers that can be expressed. If we own nine of something and add one additional object, we express the number of our possessions by changing the 9 to a 0 and adding 1 in the next place. Thus  $9 + 1 = 10$ . While adding this wrinkle to our number system extends the range of our numbers indefinitely it requires that we now keep track of the position of each of the digits. Implied in our expression of decimal numbers is the fact that there are a set of multipliers built into every number. This is apparent even in the way we read a number. Thus 375 is read as three HUNDRED and SEVENTY five. In fact we can think of 375 as  $3 \times 100 + 7 \times 10 + 5 \times 1$ . While this is not particularly earthshattering we tend to forget it which will make understanding the other number systems very difficult.

While the decimal number system is very convenient for humans it is impractical for computers which cannot operate efficiently if they have to distinguish between the ten different states or voltages that would be necessary to express a decimal number in the innards of the computer. In fact, computers have a hard time counting past one. Fortunately there is a number system invented just for this purpose. It is the binary number system which requires only two numerals, 0 and 1. This statement immediately conjures up visions of a very dumb computer doing arithmetic that could be done by any bright first-grader. This is not too far from the truth; the differ-

ence lies in the fact that the computer simply overwhelms us with speed. It can do simple additions at the rate of 200,000 a second, and with that kind of speed at our disposal even a machine operating at the  $1 + 1$  level can make short work out of some pretty hairy problems.

Unfortunately expressing long numbers in binary is about as convenient as reciting the Gettysburg Address backwards. Even simple numbers get very long and awkward; the decimal number 314 becomes 100111010. Never mind how we convert them; we'll cover that later. For now the point is simply that numbers in binary are made up only of 0's and 1's and are one heck of a lot longer than numbers written in decimal. The fact that the computer operates only in binary numbers is just going to make it tough on the human operator.

But wait! Why not write a program that takes signals from a decimal keyboard, converts them into binary for the computer operations, executes the real program getting the results in binary, and then converts the answer back into decimal so the operator can read decimal answers off the displays. That way the human only needs to worry about his problem in good, old familiar decimal; the computer takes care of converting the numbers back and forth into binary. After all, computers are supposed to be labor saving devices. Unfortunately the program for doing a decimal to binary conversion is not exactly trivial. Also many times we will want to go from binary to decimal mentally without having to tie up the computer. Let's compromise. No, the machine cannot operate in decimal; we're going to have to change the system we as humans use. What we must do is find a system other than decimal that we can convert to binary easily and still have the brevity and flexibility that we have come to expect out of the old decimal system.

The solution is called the hexadecimal system, hex for short. It makes use of sixteen symbols or numerals and has the advantage that numbers expressed in this system are even shorter than in decimal. What's more with a little practice you can do the conversion to binary and back to hexadecimal in your head.

Since the ia7301 uses hexadecimal for the keyboard and displays we can use it to demonstrate the number system. First clear the displays by pressing **DCM**. Then key in address 0000.

Enter:

**DCM 0 0 0 0 NXT**

See Displayed:

**n00000-bF**

Forgetting for a moment about the BF contents of the location 0000 we want to examine the address itself. As we have already seen, depressing **NXT** will cause the address to be incremented.

Enter:

**NXT**

**NXT**

**NXT**

**NXT**

**NXT**

**NXT**

**NXT**

**NXT**

**NXT**

See Displayed:

**n00000-bF**

**n00001-83**

**n00002-bF**

**n00003-A4**

**n00004-C0**

**n00005-C0**

**n00006-C0**

**n00007-AB**

**n00008-06**

**n00009-00**

So far we have demonstrated only that the ia7301 can count. Starting at 0000 we have successfully counted up to an astounding 0009. Now, by all that's right we should be able to press **NXT** one more time and extend the count to ten, 0010. However when we actually perform the operation, something entirely different occurs.

Enter:

See Displayed:

n00009-00

**NXT**

n0000A-10

As has been pointed out many times in the ia7301 promotional literature, this computer operates with hexadecimal keys and displays. Since the hexadecimal number system uses sixteen symbols or numerals, obviously the decimal set of 0 through 9 will not be sufficient. The discrepancy is taken care of by adding the letters A through F to the numerals 0 through 9. When we incremented the address 0009 above, the system correctly incremented it to 000A which is the number after 0009 IN THE HEXADECIMAL SYSTEM.

Now since you and I both know perfectly well that the number after 9 is ten we are led to the unmistakable conclusion that A must be the hexadecimal equivalent of decimal ten. If we keep track of the decimal equivalents as we go, we can use the ia7301 to make up a table of decimal-hexadecimal equivalents. Use the form below (we've already filled in a few of the blanks) to make up your own table. Use **NXT** key to tell you each hexadecimal number.

## DECIMAL

0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20

## HEXADECIMAL

0  
1  
\_\_\_\_\_  
\_\_\_\_\_  
4  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
9  
A  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

Through, already? Good. Your table should look like the one below. if it doesn't, go back and check your results using the ia7301. Remember to ignore everything in the displays except the address digits.

DECIMAL	HEXADECIMAL
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F
16	10
17	11
18	12
19	13
20	14

IDIOSYNCRASY: THERE IS A LIMITED NUMBER OF SYMBOLS THAT A SEVEN SEGMENT DISPLAY IS CAPABLE OF HANDLING. THE A, C, E AND F ARE FINE; BUT THE ONLY WAY WE COULD SHOW B AND D ON THE DISPLAY IS BY GOING TO THE LOWER CASE LETTERS, b AND d. THIS APPARENT ANOMALY WILL BE FORGOTTEN AFTER A FEW HOURS OF PRACTICE WITH THE COMPUTER.

There are some obvious lessons to be learned from our little table. First, there is no difference between the decimal and hexadecimal numbers as long as we stay in the 0 to 9 range. This is one of the reasons why the hexadecimal system is so easy to learn. With 16 symbols to work with instead of only 10, numbers remain in the one-digit range longer with hexadecimal than with decimal; this makes hex numbers shorter than their decimal equivalents. When we try to count beyond the range of the single digit capacity of our sixteen symbols, it becomes necessary to use a second digit, hence the change from 000F to 0010. It's easy enough to remember the hex equivalents when working with only a single digit, but many people forget when they go to two digits. 10 in hex is NOT ten or twenty but sixteen (check your chart). With some practice this tendency will pass.

Because the ia7301 keyboards and displays all operate in hexadecimal, you will probably begin to think of the computer as operating in hexadecimal. **THIS IS NOT TRUE!** The computer has, and always will, operate in binary. The displays and keys operate in hex only because the computer has a binary coded program within it that does a little translation routine at the keys and displays. The computer "thinks" in binary the same way you think in English. Granted you may take a high-school French course but the fact remains that you are thinking in English and doing mental translations to French; the computer feels the same way about binary and hexadecimal. We're going to spend a lot more time on the binary number system later in this course. Right now we're going to concentrate on hexadecimal because that's the system used to communicate with the computer.

What happens when we count past 19 on the computer? If we were in decimal the next number would be 20, but since we are using hexadecimal let's just ask the computer.

Enter: \*

DCM 0 0 1 9 NXT

NXT

NXT

NXT

NXT

NXT

NXT

NXT

See Displayed:

n0019- -

n001A- -

n001b- -

n001C- -

n001d- -

n001E- -

n001F- -

n0020- -

As you probably expected, the hex system always follows a 9 with an A, B, C, etc. Once the 1F stage is reached, incrementing the address again causes the F to be reset back to 0 and the next digit incremented to 2. What happens when 9F is incremented? Try it!

Enter:

DCM 0 0 9 F NXT

NXT

NXT

See Displayed:

n009F- -

n00A0- -

n00A1- -

and so on.

What about the situation when 00FF is incremented? By now you should know the answer yourself, but try it anyway.

Enter:

DCM 0 0 F F NXT

NXT

NXT

See Displayed:

h00FF-

h0100-

h0101-

and so on.

By now it has probably occurred to you that there is no way of telling certain decimal numbers from hexadecimal numbers. After all, it makes a big difference whether  $9 + 1 = ?$  is being expressed in decimal or hex. To keep them apart we will always follow a hex number with H. Thus  $9 + 1 = 10$  but  $9H + 1H = A$ . Since the H does not appear in the displays or on the keyboard we will not use it when picturing these.

By this time you should have a good feeling for the hexadecimal number system, so we're going to go on to how to use the computer memory. First, though, let's be sure you really now have:

**SKILL 1. YOU SHOULD BE TOTALLY COMFORTABLE IN THE HEXADECIMAL NUMBER SYSTEM. THIS MEANS THAT YOU SHOULD KNOW THE DECIMAL EQUIVALENTS OF 0-F WITHOUT HAVING TO GO THROUGH A MENTAL COUNTING EXERCISE. YOU SHOULD ALSO KNOW WHAT THE HEX NUMBERS AFTER 01FFH, 0FFFH, AND 0F9FH ARE. IF THESE ARE RELATIVELY EASY QUESTIONS, GO ON TO CHAPTER TWO. IF NOT, GO TO THE SUPPLEMENTARY EXERCISE 1 IN THE BACK OF THIS BOOK BEFORE GOING TO CHAPTER TWO.**

